Available online at www.sciencedirect.com
ScienceDirect
The Journal of Systems and Software 80 (2007) 198–215
The Journal of Systems and Software
www.elsevier.com/locate/jss

GSR: A global seek-optimizing real-time disk-scheduling algorithm

Hsung-Pin Chang [a,*], Ray-I Chang [b], Wei-Kuan Shih [c], Ruei-Chuan Chang [d]

[a] Department of Computer Science, National Chung Hsing University, 250 Kuo Kuang Road, Taichung 402, Taiwan, ROC
[b] Institute of Engineering Science and Ocean Engineering, National Taiwan University, Taipei, Taiwan, ROC
[c] Department of Computer Science, National Tsing Hau University, Hsinchu, Taiwan, ROC
[d] Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, ROC

Received 22 August 2005; received in revised form 31 March 2006; accepted 31 March 2006
Available online 22 May 2006

Abstract

Earliest-deadline-first (EDF) is good for scheduling real-time tasks in order to meet timing constraint. However, it is not good enough for scheduling real-time disk tasks to achieve high disk throughput. In contrast, although SCAN can maximize disk throughput, its schedule results may violate real-time requirements. Thus, during the past few years, various approaches were proposed to combine EDF and SCAN (e.g., SCAN-EDF and RG-SCAN) to resolve the real-time disk-scheduling problem. However, in previous schemes, real-time tasks can only be rescheduled by SCAN within a local group. Such restriction limited the obtained data throughput. In this paper, we proposed a new *globally* rescheduling scheme for real-time disk scheduling. First, we formulate the relations between the EDF schedule and the SCAN schedule of input tasks as *EDF-to-SCAN mapping* (ESM). Then, on the basis of ESM, we propose a new real-time disk-scheduling algorithm: globally seek-optimizing rescheduling (GSR) scheme. Different from previous approaches, a task in GSR may be rescheduled to anywhere in the input schedule to optimize data throughput. Owing to such a globally rescheduling characteristic, GSR obtains a higher disk throughput than previous approaches. Furthermore, we also extend the GSR to serve fairly non-real-time tasks. Experiments show that given 15 real-time tasks, our data throughput is 1.1 times that of RG-SCAN. In addition, in a mixed workload, compared with RG-SCAN, our GSR achieves over 7% improvement in data throughput and 33% improvement in average response time.

Keywords: Real-time disk scheduling; Disk scheduling; Operating systems

1. Introduction

Recent advancement in hardware technology and network communications has increased the popularity of data services. In some applications, the data services must be provided with timing characteristics. For example, media data must be accessed under real-time constraints to guarantee jitter-free playback. Furthermore, media data are often in large volume and consume significant disk bandwidth. As a result, performances of multimedia applications depend heavily on the real-time disk-scheduling algorithm applied (Lougher and Shepherd, 1993; Steinmetz, 1995). A well-behaved real-time disk scheduling should maximize data throughput while guaranteeing real-time constraints.

The earliest time at which a disk task can start is defined as its *ready time* (or *release time*). The latest time at which a disk task must be completed is its *deadline*. The actual times at which a disk task is started and completed are its *start-time* and *fulfill-time*, respectively. To meet timing constraints for real-time data services, each disk task must guarantee

* Corresponding author. Tel.: +886 4 22852106; fax: +886 4 22853869.
E-mail address: hpchang@cs.nchu.edu.tw (H.-P. Chang).

0164-1212/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2006.03.045

that its start-time scheduled is not earlier than its ready time achieved. Moreover, its fulfill-time scheduled is not later than its deadline set (Stankovic and Buttazzo, 1995). Different from the conventional disk-scheduling problem, timing constraints on accessing real-time data is crucial for supporting timing critical applications (Anderson et al., 1991, 1992; Chang et al., 1997). Although the well-known SCAN algorithm, which scans disk surface back and forth to retrieve the data under disk head, has been proved as the best algorithm for maximizing disk throughput (Chen and Yang, 1992; Chen et al., 1992), its output result may not meet timing constraint on scheduling real-time disk tasks. Therefore, real-time data retrieved by SCAN may be meaningless and even harmful to systems (Gemmell et al., 1995; Gemmell and Christodoulakis, 1992).

In contrast, earliest-deadline-first (EDF), which serves tasks in deadline order, is one of the best-known schemes for scheduling real-time tasks (Liu and Layland, 1973; Lehoczky, 1990). However, EDF earns its optimization under the assumption that tasks are independent. Nevertheless, in real-time disk scheduling, tasks are non-preemptive and inter-dependent. Once a disk task is issued to a disk drive, its service cannot be interrupted. Moreover, for each disk task, its service time depends not only on the location of the data block retrieved, but also on the location of the current disk head. As a result, taking only deadlines into account without service time consideration, EDF incurs excessive seek-time costs and results in poor disk throughput (Yee and Varaiya, 1991; Reddy and Wyllie, 1994). Actually, owing to the non-preemptive property and the non-prespecified service time of disk tasks, it is very hard to optimize the schedule result of real-time disk tasks. This problem has been proved to be NP-complete (Wong, 1980).

Previous studies have examined heuristic methods for combining the features of SCAN type of seek-optimizing algorithms with EDF type of real-time scheduling algorithms. In 1993, Reddy and Wyllie proposed the *SCAN-EDF* method that first sorts input tasks by the EDF order and, then reschedules tasks with the same deadlines by SCAN. Experiments show that their obtained results depend highly on the probability of tasks that have the same deadlines. To increase the probability of employing SCAN to reschedule tasks, DM-SCAN (deadline-modification-SCAN) and RG-SCAN (reschedulable-group-SCAN) are proposed to select automatically contiguous tasks that can be rescheduled by SCAN (Chang et al., 1998, 2002). In other words, these contiguous tasks can be viewed as having the "same deadline" in SCAN-EDF.

However, previous approaches are *locally seek-optimizing* schemes; i.e., tasks can only be rescheduled by SCAN within a local group. Note that, each group is a set of *consecutive* tasks that can be rescheduled by SCAN without missing their respective timing constraints. For example, in SCAN-EDF, a group is made up of tasks having the same deadline. Similarly, given a set of EDF tasks, DM-SCAN automatically selects groups of consecutive tasks and these groups are named as MSGs (maximum-scannable-groups). RG-SCAN also has its own group definition and is called R-Group (reschedulable-group). However, no matter in SCAN-EDF, DM-SCAN, or RG-SCAN, once a task belongs to a certain group, it cannot be rescheduled to a different group; even though such a rescheduling derives a better performance. The detailed operations of DM-SCAN and RG-SCAN with their proposed MSG and R-Group concepts are introduced in Section 2.

To resolve the drawback of previous approaches, we propose herein a *globally seek-optimizing* scheduling approach: *GSR* (globally seek-optimizing rescheduling) scheme. First, a graph of *EDF-to-SCAN mapping* (ESM) is introduced to explore relations between the EDF schedule and the SCAN schedule of input tasks. Given a set of real-time disk tasks, schedule results of EDF and SCAN just denote two permutations of input tasks. By representing each task as a vertex and connecting each task in the EDF schedule to the same task in the SCAN schedule with an edge, there is a bipartite mapping; which is called ESM in this paper. On the basis of this ESM mapping, our algorithm then identifies *scan-groups* where each scan-group contains the maximum number of contiguous tasks that are in the same SCAN direction (left-to-right or right-to-left). Now, the input schedule can be viewed as a piecewise-SCAN schedule. After that, input tasks are tested for being rescheduled into suitable scan-groups to achieve the highest improvement of disk throughput while guaranteeing real-time requirements. Thus, our scheme provides a good combination of the EDF scheme and the SCAN scheme.

Note that, since there are at most n scan-groups where n is the number of input tasks, a naive algorithm will take $O(n^2)$ time to decide the best reschedule result for the selected task. To speed up its computation, we introduce a concept of the *schedulable-region* to each input task. With the help of the pre-computed schedulable-regions, the best-fit scan-group for rescheduling each input task can be decided in $O(n)$ time. In addition, we extend the GSR to serve mixed real-time/non-real-time disk tasks such that non-real-time tasks can be served to minimize response time while guaranteeing the timing constraints of real-time tasks. Compared with DM-SCAN, experiments show that our GSR algorithm can support over 11% data throughput improvement in a real-time system. Moreover, in a mixed workload, our GSR achieves over 7% improvement compared with RG-SCAN scheme in data throughput and offers 33% improvement compared with RG-SCAN in terms of average response time of non-real-time tasks.

The remainder of this paper is organized as follows. Section 2 gives mathematical definitions about real-time disk scheduling and shows some related work. The EDF-to-SCAN mapping and our proposed GSR algorithm are introduced in Section 3. In Section 4, we present the definition of reschedulable region and proposed a speed-up method for scheduling. Section 5 demonstrates how GSR is extended to efficiently serve mixed real-time/non-real-time disk tasks. Finally, Sections 6 and 7 show the experimental results and conclusion remarks, respectively.

2. Problem descriptions and related work

2.1. Real-time disk-scheduling problem

The problem input considered in this paper is a set of real-time disk tasks $\boldsymbol{T} = \{T_0, T_1, \ldots, T_n\}$ where n is the number of tasks. The ith task is represented by $T_i = (r_i, d_i, a_i, l_i, b_i)$ where r_i is its ready time, d_i is its deadline, a_i is its track location, l_i is its sector number and b_i is its data size. While serving disk task T_i, the disk head needs to be moved from the current track to the target track a_i by a *seek time* cost. Then, a rotational latency is presented for the desired sector l_i rotated under the disk head. Finally, data under disk head are retrieved with size b_i by a transfer time. The first task T_0 is assigned as a special task to represent the initial location of disk head. Without loss of generality, it can be assumed to be at the outermost track (track 0). Assume that the schedule sequence is T_jT_i (T_i is served after T_j). The service time of task T_i is calculated as,

$$c_{j,i} = seek\_time(\boldsymbol{abs}(a_i - a_j)) + rotational\_latency(l_i) + transfer\_time(b_i). \tag{1}$$

Clearly, the service time not only depends on the issued disk task itself, but is also related to the previous one. For example, in a HP 97560 hard disk (Ruemmler and Wilkes, 1994), the seek time $seek\_time_{j,i}$ with moving distance $D_{j,i} = |a_j - a_i|$ can be modeled by

$$seek\_time_{j,i} = \begin{cases} 3.24 + 0.4\sqrt{D_{j,i}}, & D_{j,i} \leqslant 383, \\ 8.00 + 0.008D_{j,i}, & D_{j,i} > 383. \end{cases} \tag{2}$$

Since T_i is a real-time task, the ready time r_i and deadline d_i are used to characterize its timing constraint (Stankovic and Buttazzo, 1995). Because disk service is non-preemptive, the related start-time and fulfill-time are $e_i = \max\{r_i, f_j\}$ and $f_i = e_i + c_{j,i}$. A simple example $\boldsymbol{T} = \{T_0, T_1, T_2, T_3\}$ for demonstrating the terminology used in this paper is shown in Fig. 1.

A schedule result of real-time disk tasks $\boldsymbol{T} = \{T_0, T_1, \ldots, T_n\}$ is called *feasible* if all input tasks T_i, for $i = 0$ to n, satisfy real-time requirements $r_i \leqslant e_i$ and $f_i \leqslant d_i$ (Lehoczky, 1990; Stankovic and Buttazzo, 1995). To measure the efficiency of a real-time disk scheduling algorithm, given a set of real-time disk tasks, the applied disk scheduling algorithm should serve as many tasks as possible under tasks' timing constraints. If the same number of tasks is feasibly served, the applied disk scheduling algorithm needs to maximize data throughput.

To determine the data throughput improvement, we define *schedule fulfill-time* as the finish time it takes to serve all input tasks according to their respective timing constraints. Clearly, this is the finish time of the latest task $f_{(n)}$. Since the disk throughput is related to the inverse of schedule fulfill-time, thus, the problem objective to maximize disk throughput can be redefined as to minimize schedule fulfill-time. In real-time disk scheduling, the system time required for serving each task is determined by its schedule sequence (Peterson and Silberschatz, 1985). However, the schedule sequence of a task depends on its service time required. Thus, it is hard to decide the optimal schedule result that maximizes the disk throughput without violating real-time requirements.

2.2. Related work

Owing to the NP-complete feature, previous real-time disk scheduling algorithms thus apply heuristically the seek-optimizing SCAN scheme to an EDF schedule for reducing the disk service time. For example, the well-known SCAN-EDF scheme reschedules tasks having the same deadline in an EDF order to reduce service times of tasks. However, since only

request	r_i	d_i	a_i	b_i
T_0	0	0	0	0
T_1	1	11	2	1
T_2	0	5	4	1
T_3	3	12	5	1

$c_{j,i}$	$i = 0$	$i = 1$	$i = 2$	$i = 3$
$j = 0$	-	3	5	6
$j = 1$	3	-	3	4
$j = 2$	5	3	-	2
$j = 3$	6	4	2	-

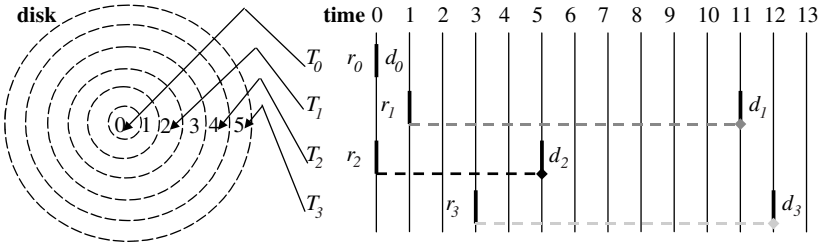

Fig. 1. The terminology used for a simple example $\boldsymbol{T} = \{T_0, T_1, T_2, T_3\}$. For simplicity, we ignore the rotational latency.

tasks having the same deadline are seek-optimized, the reduction in schedule fulfill-time compared with EDF is not significant. To increase the probability of employing the SCAN scheme to reschedule input tasks, DM-SCAN (deadline-modification-SCAN) proposed the concept of maximum-scannable-group (MSG) (Chang et al., 1998). Given an EDF schedule, *consecutive* tasks that can be rescheduled by SCAN without missing their respective timing constraints can be directly derived by the concept of MSG. Given a set of real-time disk tasks with EDF-ordered $\boldsymbol{T} = T_1T_2\ldots T_n$, the MSG G_i starting from T_i is defined as the sequential tasks $G_i = T_iT_{i+1}T_{i+2\ldots}T_{i+m}$ with each task T_k for $k = i$ to $i+m$ satisfies $f_k \leqslant d_i$ and $r_k \leqslant s_i$. However, DM-SCAN requires that the input tasks must be EDF-ordered. Therefore, they proposed a *deadline-modification* scheme that transfers a non-EDF schedule into an EDF order by modifying tasks' deadlines. Unfortunately, in order to guarantee real-time constraints, the modified deadlines are earlier than the original ones. As a result, the deadline modification scheme causes a negative impact on the number of supported tasks by DM-SCAN.

To relieve from such a constraint, RG-SCAN (reschedulable-group-SCAN) is proposed with the concept of R-Group (reschedulable group). Given a set of real-time disk tasks $\boldsymbol{T} = T_1T_2\ldots T_n$, the R-Group G_i starting from task T_i is defined as the maximum number of consecutive tasks $G_i = T_iT_{i+1}\ldots T_{i+m}$ with each task T_k for $k = i$ to $i+m$ satisfies following criteria:

$$f_{i+m} \leqslant \min_{k=i}^{i+m}\{d_k\} \quad \text{and} \quad \max_{k=i}^{i+m}\{r_k\} \leqslant s_i. \tag{3}$$

RG-SCAN also shows that after seek-optimizing tasks within an R-Group, it can obtain more data throughput while guaranteeing real-time requirements.

However, previous approaches are locally seek-optimizing algorithms. SCAN scheme is only applied to a set of *consecutive* tasks. Thus, a task would only be rescheduled by SCAN within its own group. In other words, a task belonging to a group i, say R-Group, cannot be rescheduled to another group j, since this would violate the constraints of Eq. (3) even if such a rescheduling would derive a higher disk throughput. Therefore, in this paper, we propose a globally seek-optimizing real-time disk-scheduling algorithm, GSR, to overcome the limitations of previous approaches. In GSR, a task belonging to a group i would be "globally" rescheduled to another group j as long as the new rescheduled result obtains a higher data throughput while guaranteeing real-time constraints.

3. GSR: Globally seek-optimizing rescheduling scheme

In this section, we present the design of our proposed GSR real-time disk-scheduling algorithm. Section 3.1 first shows the construction scheme of *EDF-to-SCAN mapping* (ESM) graph to relate an EDF schedule to a SCAN schedule. Then, we present the idea of *scan-groups*, which is derived from an ESM graph. On the basis of scan-groups, our proposed GSR algorithm is described in Section 3.2.

3.1. EDF-to-SCAN mapping

As described in Section 1, although SCAN can maximize data throughput, its schedule result does not meet the timing constraints of real-time tasks. In contrast, the EDF schedule is good for real-time requirements. However, its disk throughput is low. Fig. 2 demonstrates the SCAN schedule and EDF schedule of the example shown in Fig. 1. In Fig. 2(a), the

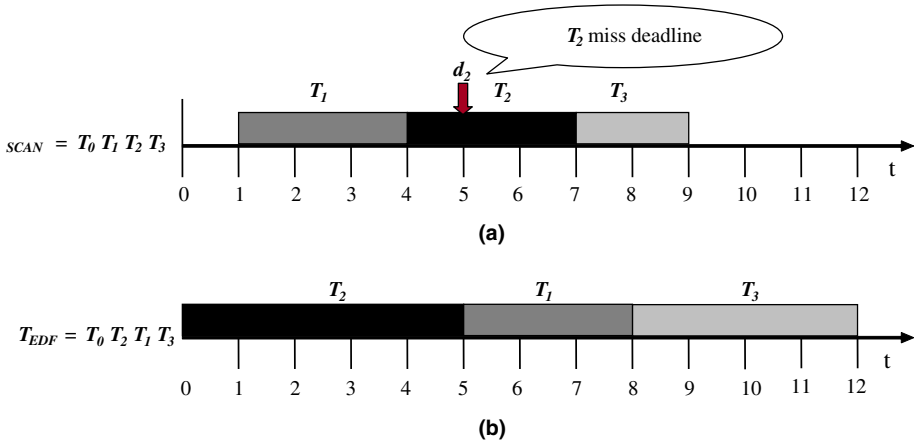

Fig. 2. For the example presented in Fig. 1, the schedule results obtained by (a) the SCAN approach and (b) the EDF approach are demonstrated. Note that the schedule result obtained by SCAN is not feasible.

SCAN schedule derives a shorter schedule fulfill-time but is not feasible. In contrast, the EDF schedule shown in Fig. 2(b) owns a feasible result but results in a longer schedule-fulfill time. There is a tradeoff relation between these two extreme schedule cases. It motivates us to construct a graph of *EDF-to-SCAN mapping* (ESM) to develop a new scheme for real-time disk scheduling.

Given a set of real-time disk tasks $\boldsymbol{T} = \{T_0, T_1, \ldots, T_n\}$, their SCAN schedule is $\boldsymbol{T}_{\mathrm{SCAN}} = T_{S(0)}T_{S(1)}\ldots T_{S(n)}$ with data locations $a_{S(0)} \leqslant a_{S(1)} \leqslant \cdots \leqslant a_{S(n)}$ where $S(i)$, for $i = 0$ to n, is a permutation of indexes $\{0, 1, \ldots, n\}$. We can represent $\boldsymbol{T}_{\mathrm{SCAN}}$ by an one-dimensional graph $G_{\mathrm{SCAN}} = (V_{\mathrm{SCAN}}, E_{\mathrm{SCAN}})$ where the vertex set $V_{\mathrm{SCAN}} = \boldsymbol{T}$ and the edge set $E_{\mathrm{SCAN}} = \{(T_{S(i-1)}, T_{S(i)}) \mid \text{for } i = 1 \text{ to } n\}$. The same idea can be applied to tasks' EDF schedule $\boldsymbol{T}_{\mathrm{EDF}} = T_{E(0)}T_{E(1)}\ldots T_{E(n)}$ with deadlines $d_{E(0)} \leqslant d_{E(1)} \leqslant \cdots \leqslant d_{E(n)}$ where $E(i)$, for $i = 0$ to n, is also a permutation of indexes $\{0, 1, \ldots, n\}$. The related graph is $G_{\mathrm{EDF}} = (V_{\mathrm{EDF}}, E_{\mathrm{EDF}})$, where the vertex set $V_{\mathrm{EDF}} = \boldsymbol{T}$ and the edge set $E_{\mathrm{EDF}} = \{(T_{E(i-1)}, T_{E(i)}) \mid \text{for } i = 1 \text{ to } n\}$. Since $V_{\mathrm{SCAN}} = V_{\mathrm{EDF}} = \boldsymbol{T}$, there is a bipartite mapping between G_{SCAN} and G_{EDF}.

Definition 1. [*EDF-to-SCAN mapping* (*ESM*)] EDF-to-SCAN mapping of real-time disk tasks $\boldsymbol{T} = \{T_0, T_1, \ldots, T_n\}$ is a bipartite graph $G_{\mathrm{ESM}} = (V_{\mathrm{ESM}}, E_{\mathrm{ESM}})$ that satisfies (1) the vertex set $V_{\mathrm{ESM}} = V_{\mathrm{EDF}} \cup V_{\mathrm{SCAN}}$, and (2) the edge set $E_{\mathrm{ESM}} = \{(T_{E(j)}, T_{S(i)}) \mid \text{for } T_{E(j)} \in V_{\mathrm{EDF}}, T_{S(i)} \in V_{\mathrm{SCAN}} \text{ and } T_{E(j)} = T_{S(i)}\}$.

Fig. 3(a) shows an example of ESM for the input tasks in Fig. 1. Using this mapping, we can investigate the possible transformations from the real-time EDF schedule to the seek-optimized SCAN schedule to find a good real-time disk schedule.

To mimic the behavior of the SCAN schedule, we first track the scan directions of tasks in the input schedule (EDF, usually but not necessary) to decompose the input schedule into a sequence of *scan-groups* where each scan-group contains the maximum number of contiguous tasks with the same SCAN direction. Therefore, the input schedule can be represented by a *piecewise-SCAN* schedule. Given an input schedule $T_0T_1T_2\ldots T_n$, an O(n) algorithm to identify all these scan-groups is shown as follows:

Algorithm 1 (*Scan-groups identification* (*SGI*))

```
/* INPUT: an EDF schedule T0T1T2...Tn. OUTPUT: a set of scan-groups. */
Initial the 1-st scan-group S1 = T0T1;
Initial direction = +1;   /* from location a0 = 0 to location a1, where a0 ⩽ a1 */
i = 1;                    /* the index of scan-group */
for k = 2 to n do begin   /* for each task */
   if (a_{k-1} ⩽ a_k) then new_direction = +1;
                      else new_direction = −1;
   if (direction = new_direction) then S_i = S_i + T_k; /* in the same scan-group */
                                  else begin            /* new scan-group */
                                         i = i + 1;
```

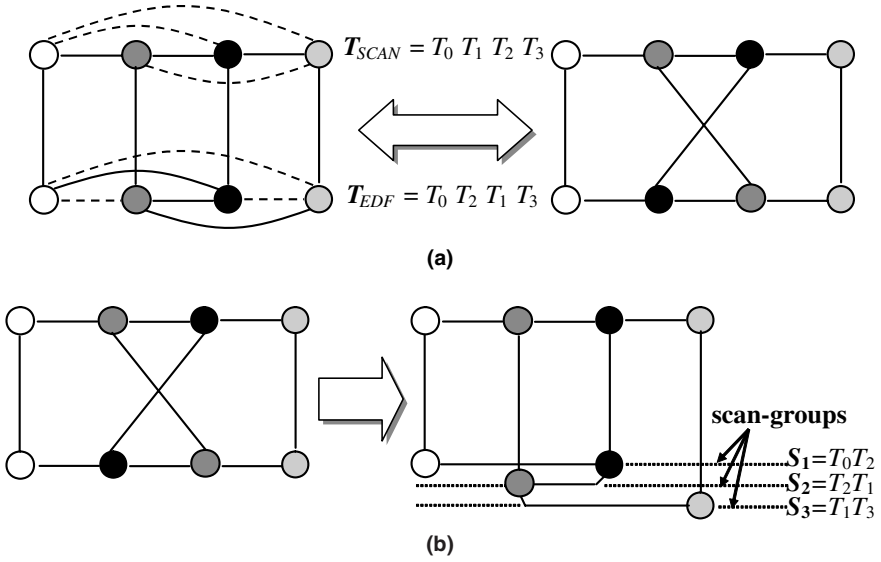

Fig. 3. (a) The EDF-to-SCAN mapping (ESM) graph for the tasks presented in Fig. 1 is shown. (b) Based on this mapping graph, we can identify scan-groups for the EDF schedule.

Initial the ith scan-group $\boldsymbol{S}_i = T_{k-1}T_k$;
Initial *direction* = *new_direction*;
end /* else */
end /* for */

As the example shown in Fig. 3(b), by mapping the input EDF schedule $T_0T_2T_1T_3$ to the SCAN schedule $T_0T_1T_2T_3$, we obtain a piecewise-SCAN schedule with three scan-groups $\boldsymbol{S}_1 = T_0T_2$, $\boldsymbol{S}_2 = T_2T_1$ and $\boldsymbol{S}_3 = T_1T_3$. It introduces a new point of view for analyzing the relations between the input schedule and the SCAN schedule. Different heuristic methods can be developed for real-time disk scheduling. For example, we may try to minimize the number of scan-groups or to maximize the sizes of scan-groups under real-time requirements of such a piecewise-SCAN schedule. Since the more the tasks can be seek-optimized, the more the disk throughput is obtained. In this paper, on the basis of ESM, an effective and efficient real-time disk scheduling method GSR is proposed. Note that although the same idea can be employed to represent a SCAN schedule by a *piecewise-EDF* schedule, its schedule result usually violates real-time requirements.

3.2. GSR algorithm

In this subsection, we describe our proposed GSR algorithm. Without loss of generality, we assume that the input is a feasible $\boldsymbol{T}_{\mathrm{EDF}}$ schedule to meet basic timing constraints. Our algorithm then selects an input task according to the first-in-first-serve (FIFS) order and tries to reschedule the selected task into the *best-fit* scan-group to maximize disk throughput under real-time requirements. For example, given the input tasks shown in Fig. 1, we can improve disk throughput by rescheduling task T_3 into scan-group T_0T_2. The new scan-groups are $\boldsymbol{S}_1 = T_0T_2T_3$ and $\boldsymbol{S}_2 = T_3T_1$, as shown in Fig. 4. Note that, it reduces the number of scan-groups by 1 ($\boldsymbol{S}_3$ is removed) and increases the size of scan-group $\boldsymbol{S}_1$ from 2 to 3. In this paper, a rescheduled result is accepted only when it is feasible and the disk throughput is improved. Considering the input tasks shown in Fig. 4, T_2 will miss its deadline if T_1 is rescheduled into the scan-group T_0T_2. The rescheduled result $T_0T_1T_2T_3$ is not acceptable as it violates real-time constraints. A detailed description of the proposed GSR algorithm is illustrated as follows.

Algorithm 2 (*GSR real-time disk-scheduling algorithm*)

/*INPUT: an feasible EDF schedule $T_0T_1T_2\ldots T_n$. OUTPUT: an improved schedule result.*/
Identify all scan-groups $\{\boldsymbol{S}_1, \boldsymbol{S}_2, \ldots\}$ of input schedule $T_0T_1T_2\ldots T_n$;
for $i = 2$ **to** n **do begin** /* for all tasks T_i */
  Assume that T_i is in the jth scan-group $\boldsymbol{S}_j$;
  Initialize the index of scan-group tested by task T_i as $p = j$;
  The initial value of the improvement of data throughput is $\boldsymbol{O}_p = 0$;
  for $q = j$-1 **down to** 1 **do begin** /* test all scan-groups S_j */
    Try to reschedule task T_i into scan-group S_q as a new schedule;
    Compute the improvement of data throughput $\boldsymbol{O}_q$ after rescheduling;
    if ((the new schedule is feasible) and ($\boldsymbol{O}_p \leqslant \boldsymbol{O}_q$))
      then the new index of the best-fit scan-group is $p = q$;

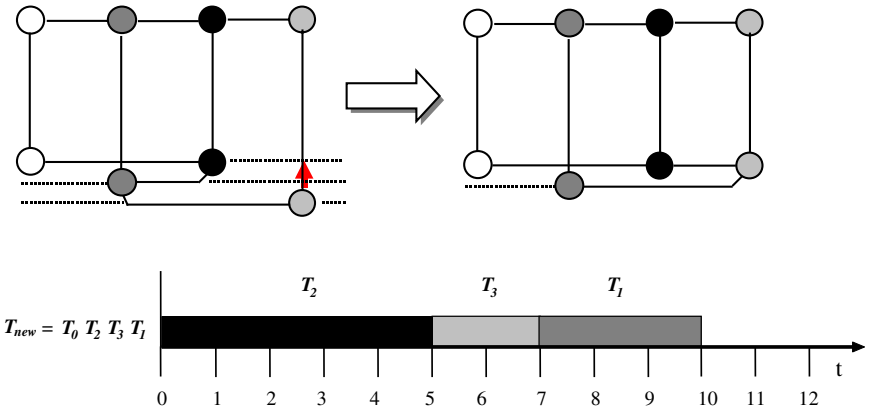

Fig. 4. For the example presented in Fig. 1, we can select a suitable task T_3 and reschedule it into the suitable scan group T_0T_2 to improve the disk throughput.

```
    end /* for */

  if (O_p > 0) then do begin
     Reschedule T_i into S_p as the rescheduled result T_resch
     Identify all scan-groups {S_1, S_2, ...} of T_resch;
  end /* if */
end /* for */
```

After identifying all scan-groups, GSR tries to reschedule each task into all scan-groups before its own scan-group and to compute the improvement of data throughput of each rescheduling result. The one with the largest throughput improvement while guaranteeing a feasible schedule is selected for rescheduling (the task is rescheduled from its own scan-group to the new one.) Notably, the above algorithm assumes that the input schedule is feasible. However, even an infeasible input schedule is given, GSR may still produce feasible output schedule, although it is not guaranteed.

Example 3.1. We show an example to clarify the GSR algorithm. Fig. 5 shows a set of five tasks with their timing attributes, track numbers, and the requested data size. T_0 is added as a special task to represent the initial location of the disk head and is assumed to be at track 0.

(1) Suppose that the input schedule is EDF-ordered and $\boldsymbol{T}_{\text{EDF}} = \boldsymbol{T}_0\boldsymbol{T}_2\boldsymbol{T}_1\boldsymbol{T}_3\boldsymbol{T}_4\boldsymbol{T}_5$. After applying Algorithm 1, we have three scan groups: $\boldsymbol{S}_1 = \boldsymbol{T}_0\boldsymbol{T}_2$, $\boldsymbol{S}_2 = \boldsymbol{T}_2\boldsymbol{T}_1$, and $\boldsymbol{S}_3 = \boldsymbol{T}_3\boldsymbol{T}_4\boldsymbol{T}_5$. Furthermore, from Fig. 5, the schedule fulfill time of $\boldsymbol{T}_{\text{EDF}} = 19$.
(2) $i = 2$: since T_1 is in scan-group $\boldsymbol{S}_2$, thus $p = j = 2$, $\boldsymbol{O}_p = 0$.
  [1] $q = 1$, reschedule T_1 into scan-group $\boldsymbol{S}_1$ and the new schedule $\boldsymbol{T}_{\text{new}} = \boldsymbol{T}_0\boldsymbol{T}_1\boldsymbol{T}_2\boldsymbol{T}_3\boldsymbol{T}_4\boldsymbol{T}_5$. The schedule fulfill time of $\boldsymbol{T}_{\text{new}} = 16$. Since the new schedule is feasible and $\boldsymbol{O}_q = 21\%. > 0$, thus $p = 1$. *GSR has found a new reschedule result which is better than the input schedule.*
  [2] Since $\boldsymbol{O}_p. = 21\% > 0$, the new schedule $\boldsymbol{T}_{\text{resch}} = \boldsymbol{T}_0\boldsymbol{T}_1\boldsymbol{T}_2\boldsymbol{T}_3\boldsymbol{T}_4\boldsymbol{T}_5$. Furthermore, the new scan groups of $\boldsymbol{T}_{\text{resch}}$: $\boldsymbol{S}_1 = \boldsymbol{T}_0\boldsymbol{T}_1\boldsymbol{T}_2\boldsymbol{T}_3$, $\boldsymbol{S}_2 = \boldsymbol{T}_3\boldsymbol{T}_4$, and $\boldsymbol{S}_3 = \boldsymbol{T}_4\boldsymbol{T}_5$.
(3) $i = 3$: since T_3 is in scan-group $\boldsymbol{S}_1$, thus $p = j = 1$, $\boldsymbol{O}_p = 0$ Because $q = 0 < 1$, the second *for* loop is not executed.
(4) $i = 4$: since T_4 is in scan-group $\boldsymbol{S}_2$, thus $p = j = 2$, $\boldsymbol{O}_p = 0$.
  [1] $q = 1$, reschedule T_4 into scan-group $\boldsymbol{S}_1$ and the new schedule $\boldsymbol{T}_{\text{new}} = \boldsymbol{T}_0\boldsymbol{T}_1\boldsymbol{T}_4\boldsymbol{T}_2\boldsymbol{T}_3\boldsymbol{T}_5$. The schedule fulfill time of $\boldsymbol{T}_{\text{new}} = 13$. However, the new schedule is infeasible since f_2 (=8) $> d_2$ (=6). Actually, the $\boldsymbol{T}_{\text{new}}$ schedule is SCAN-ordered. Although it obtains the largest data throughput, however, it violates tasks' timing constraints and is not acceptable.
(5) $i = 5$: since T_5 is in scan-group $\boldsymbol{S}_3$, thus $p = j = 3$, $\boldsymbol{O}_p = 0$.
  [1] $q = 2$, reschedule T_5 into scan-group $\boldsymbol{S}_2$ and the new schedule $\boldsymbol{T}_{\text{new}} = \boldsymbol{T}_0\boldsymbol{T}_1\boldsymbol{T}_2\boldsymbol{T}_4\boldsymbol{T}_3\boldsymbol{T}_5$. The schedule fulfill time of $\boldsymbol{T}_{\text{new}} = 14$. Since the new schedule is feasible and $\boldsymbol{O}_{\text{q}} = 12.5\%. > 0$, thus $p = 2$. *GSR has found a new reschedule result which is better than the previous reschedule result.*
  [2] $q = 1$, reschedule T_5 into scan-group $\boldsymbol{S}_1$ and the new schedule $\boldsymbol{T}_{\text{new}} = \boldsymbol{T}_0\boldsymbol{T}_1\boldsymbol{T}_2\boldsymbol{T}_4\boldsymbol{T}_3\boldsymbol{T}_5$. The schedule fulfill time of $\boldsymbol{T}_{\text{new}} = 15$. The new schedule is feasible and $\boldsymbol{O}_{\text{q}} = 6.25\% > 0$. However, $\boldsymbol{O}_q < \boldsymbol{O}_p$, Thus, p is not updated.
  [3] Since $\boldsymbol{O}_p. = 12.5\% > 0$, the new schedule $\boldsymbol{T}_{\text{resch}} = \boldsymbol{T}_0\boldsymbol{T}_1\boldsymbol{T}_2\boldsymbol{T}_4\boldsymbol{T}_3\boldsymbol{T}_5$. Furthermore, the new scan groups of $\boldsymbol{T}_{\text{resch}}$: $\boldsymbol{S}_1 = \boldsymbol{T}_0\boldsymbol{T}_1\boldsymbol{T}_2$, $\boldsymbol{S}_2 = \boldsymbol{T}_2\boldsymbol{T}_4$, and $\boldsymbol{S}_3 = \boldsymbol{T}_4\boldsymbol{T}_3\boldsymbol{T}_5$.

From Example 3.1, the input schedule has its schedule fulfill time as 19. After the completion of Algorithm 2, GSR derives a reschedule result that has a schedule fulfill time of 15. As a result, compared with the input schedule, the data throughput of the new reschedule has an improvement of 21% compared to the original input schedule.

task	r_i	d_i	a_i	b_i
T_0	0	0	0	0
T_1	1	11	2	1
T_2	0	7	4	1
T_3	3	12	5	1
T_4	3	14	3	1
T_5	4	15	6	1

$c_{j,i}$	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
$j = 0$	-	3	5	6	4	7
$j = 1$	3	-	3	4	2	5
$j = 2$	5	3	-	2	2	3
$j = 3$	6	4	2	-	3	2
$j = 4$	4	2	2	3	-	4
$j = 5$	7	5	3	2	4	-

Fig. 5. The four tuple (r_i, d_i, a_i, b_i) of a set of five tasks used in Example 3.1. Furthermore, we also shows $c_{j,i}$ for all combination of schedule sequence T_jT_i. For simplicity, we ignore the rotational latency.

4. Speed-up method

However, in Algorithm 2, when rescheduling a task into a tested scan-group, a native algorithm will take O(n) time to verify the feasibility of rescheduled result and to measure the improvement of data throughput. Since there are at most n scan-groups to be tested, it totally takes O(n^2) time to decide the best-fit scan-group for each task. To accelerate the testing process, in this paper, we further introduce the *schedulable-region* concept to reduce the time complexity from O(n^2) to O(n).

In this section, we show how to speed up the testing process by the concept of *schedulable-region*. Section 4.1 should first introduce the definition of schedulable-region. After that, a fast algorithm involving the schedulable-region is presented in Section 4.2.

4.1. Schedulable-region

Before defining the schedulable-region, for each task T_k in the input schedule $T_0T_1\ldots T_n$, we first introduce the *minimal schedulable fulfill-time* f_k^{L}, the *minimal schedulable start-time* e_k^{L}, the *maximal schedulable fulfill-time* f_k^{R} and the *maximal schedulable start-time* e_k^{R}. The superscripts "L" and "R" represent "Left-most" and "Right-most", respectively.

Definition 2. [*Minimal schedulable start-time/fulfill-time*] For each task in the input schedule, the minimal schedulable start-time/fulfill-time is the earliest (Left-most) start-time/fulfill-time to serve the task without violating real-time requirements.

Given $e_0^{\mathrm{L}} = f_0^{\mathrm{L}} = 0$ to represent the initial disk head by $T_0 = (0,0,0,0,0)$, the minimal schedulable start-time e_k^{L} and the minimal schedulable fulfill-time f_k^{L} of task T_k, for $k = 1$ to n, can be computed by the *aggressive scheduling scheme* (Chang et al., 1997) as follows:

$$\begin{aligned} e_k^{\mathrm{L}} &= \max\{r_k,\ f_{k-1}^{\mathrm{L}}\}, \\ f_k^{\mathrm{L}} &= e_k^{\mathrm{L}} + c_{k-1,k}. \end{aligned} \tag{4}$$

Obviously, the start-time e_k^{L} and fulfill-time f_k^{L} obtained are minimized to serve task T_k as early as possible under a feasible schedule sequence $T_0T_1\ldots T_n$. (The real-time requirements $f_k^{\mathrm{L}} \leqslant d_k$, for $k = 1$ to n, are guaranteed.) According to the similar idea, we can define the maximal schedulable start-time and fulfill-time as follows.

Definition 3. [*Maximal schedulable start-time/fulfill-time*] For each task in an input schedule, the maximal schedulable start-time/fulfill-time is the latest (Right-most) start-time/fulfill-time for serving the task without violating real-time requirements.

According to the above definition, tasks are served as late as possible under a feasible schedule sequence $T_0T_1\ldots T_n$. To guarantee a feasible schedule, for the last task T_n, its maximal schedulable fulfill-time is its deadline, i.e., $f_n^{\mathrm{R}} = d_n$. Then, its maximal schedulable start-time can be decided by $e_n^{\mathrm{R}} = f_n^{\mathrm{R}} - c_{n-1,n} = d_n - c_{n-1,n}$. Using the same method, the maximal schedulable start-time and fulfill-time of task T_k, for $k = n-1$ down to 0, can be computed by the *lazy scheduling scheme* (Chang et al., 1997) as follows:

$$\begin{aligned} f_k^{\mathrm{R}} &= \min\{d_k,\ e_{k+1}^{\mathrm{R}}\}, \\ e_k^{\mathrm{R}} &= f_k^{\mathrm{R}} - c_{k-1,k}, \end{aligned} \tag{5}$$

where the service time $c_{-1,0}$ is assigned as 0. From Eqs. (4) and (5), $f_k^{\mathrm{L}} - e_k^{\mathrm{L}} = f_k^{\mathrm{R}} - e_k^{\mathrm{R}} = c_{k-1,k}$. Furthermore, we can prove that $e_k^{\mathrm{L}} \leqslant e_k^{\mathrm{R}}$, $f_k^{\mathrm{L}} \leqslant f_k^{\mathrm{R}}$.

Lemma 1. *By the definition of* e_k^{L}, e_k^{R}, f_k^{L}, f_k^{R} *from Eqs.* (4) *and* (5), *we have* $e_k^{\mathrm{L}} \leqslant e_k^{\mathrm{R}}$, $f_k^{\mathrm{L}} \leqslant f_k^{\mathrm{R}}$.

Proof. We prove it by the induction scheme.

(1) When $k = 1$, $e_1^{\mathrm{L}} = \max\{r_1, f_0^{\mathrm{L}}\} = r_1$ (since $f_0^{\mathrm{L}} = 0$). Thus, $f_1^{\mathrm{L}} = e_1^{\mathrm{L}} + c_{0,1} = r_1 + c_{0,1}$. In order to guarantee a feasible schedule, $r_1 + c_{0,1}$ must be smaller than or equal to d_1. Thus

$$f_1^{\mathrm{L}} = r_1 + c_{0,1} \leqslant d_1. \tag{6}$$

From Eq. (5), $f_1^{\mathrm{R}} = \min\{d_1, e_2^{\mathrm{R}}\}$, we discuss it in the following two cases.

(a) Suppose $d_1 \leqslant e_2^{\mathrm{R}}$, then, $f_1^{\mathrm{R}} = d_1$. Thus, from Eq. (6),

$$f_1^{\mathrm{L}} = r_1 + c_{0,1} \leqslant d_1 = f_1^{\mathrm{R}}. \tag{7}$$

From Eqs. (5) and (6), $e_1^R = f_1^R - c_{0,1} = d_1 - c_{0,1}$. To guarantee a feasible schedule, $d_1 - c_{0,1}$ must be larger than or equal to r_1. Thus,

$$e_1^R = f_1^R - c_{0,1} = d_1 - c_{0,1} \geqslant r_1 = e_1^L. \tag{8}$$

From Eqs. (7) and (8), we have $e_1^L \leqslant e_1^R$, $f_1^L \leqslant f_1^R$.

(b) Suppose $e_2^R \leqslant d_1$, then, $f_1^R = e_2^R$. The proof can be derived in the same way as the proof in (a)

From (a) and (b), we derive that when $k = 1$, we have $e_1^L \leqslant e_1^R$, $f_1^L \leqslant f_1^R$.

(2) Suppose when $k = n$, we have $e_k^L \leqslant e_k^R$, $f_k^L \leqslant f_k^R$. Then, when $k = n + 1$, we can prove that $e_{k+1}^L \leqslant e_{k+1}^R$, $f_{k+1}^L \leqslant f_{k+1}^R$ in the same way as the proof in (1). As a result, from (1) and (2), we prove that $e_k^L \leqslant e_k^R$, $f_k^L \leqslant f_k^R$. □

After the definition of minimal schedulable start-time/fulfill-time and minimal schedulable start-time/fulfill-time, an arbitrary *schedulable-region* $\boldsymbol{R}_{i,j}$, for $0 \leqslant i \leqslant j \leqslant n$, can be defined as follows.

Definition 4. [*Schedulable-region*] For a set of contiguous tasks $T_iT_{i+1}\ldots T_j$ in the input schedule $T_0T_1\ldots T_n$, the schedulable-region $R_{i,j}$ is the time region that $T_iT_{i+1}\ldots T_j$ can be served without violating real-time requirements.

From the definitions of the minimal/maximal schedulable start-time/fulfill-time, the schedulable-region of tasks $T_iT_{i+1}\ldots T_j$ is from the minimal schedulable start-time e_i^L of the first task T_i to the maximal schedulable fulfill-time f_j^R of the last task T_j. It is denoted as $\boldsymbol{R}_{i,j} = [e_i^L, f_j^R]$.

4.2. A fast algorithm

Assume that the task T_x $(0 < x \leqslant n)$ of input schedule $\boldsymbol{T} = T_0T_1\ldots T_n$ is selected for rescheduling. To record $\boldsymbol{T}$'s schedule fulfill-time, we add a null task T_{n+1} with $r_{n+1} = 0$ and $d_{n+1} = f_n^L$ to the end of schedule. The service time $c_{i,n+1} = 0$ for all i. Removing the selected task T_x from the input schedule, the new schedule is $\boldsymbol{T}_A = T_{A(0)}T_{A(1)}\ldots T_{A(n)} = T_0T_1\ldots T_{x-1}T_{x+1}\ldots T_{n+1}$. By applying Eqs. (4) and (5), we can pre-compute the minimal schedulable fulfill-time $f_{A(j)}^L$ and the maximal schedulable fulfill-time $f_{A(j)}^R$, for $j = 0$ to n, in O(n) time. Note that the schedulable-region of $\boldsymbol{T}_A = T_{A(0)}T_{A(1)}\ldots T_{A(n)}$, i.e., $\boldsymbol{R}_{A(0),A(n)}$ should be upper bounded by the original schedule fulfill-time f_n^L. That is, the schedulable-region $\boldsymbol{R}_{A(0),A(n)} = [0, f_{A(n)}^R = f_n^L]$. As shown in Section 3.2, the GSR tries to reschedule task T_x into different scan-groups. Assume that T_x is rescheduled into a scan-group and is served before $T_{A(k)}$. The new start-time and fulfill-time of task T_i, for $i = 0$ to $n + 1$, in the new rescheduled result $T_{A(0)}T_{A(1)}\ldots T_{A(k-1)}T_xT_{A(k)}\ldots T_{A(n)}$ are denoted as $e_i^{A(k)}$ and $f_i^{A(k)}$, respectively. Since the original schedulable region $\boldsymbol{R}_{A(k),A(k)}$ is known and the new schedulable region of $T_xT_{A(k)}$ can be decided in O(1) time, the feasibility of rescheduled result can be verified in O(1) time.

Theorem 1. *By applying the pre-computed minimal schedulable fulfill-time $f_{A(j)}^L$ and maximal schedulable fulfill-time $f_{A(j)}^R$ (for $j = 0$ to n) for input schedule $T_{A(0)}T_{A(1)}\ldots T_{A(n)}$, the feasibility of schedule $T_{A(0)}T_{A(1)}\ldots T_{A(k-1)}T_xT_{A(k)}\ldots T_{A(n)}$ can be verified in O(1) time.*

Proof. Divide the rescheduled result into three sub-schedules: $T_{A(0)}T_{A(1)}\ldots T_{A(k-1)}$, $T_xT_{A(k)}$ and $T_{A(k+1)}T_{A(k+2)}\ldots T_{A(n)}$.

(a) We can serve sub-schedule $T_{A(0)}T_{A(1)}\ldots T_{A(k-1)}$ by the aggressive scheduling scheme as shown in Eq. (3). Since $T_{A(0)}T_{A(1)}\ldots T_{A(k-1)} = T_0T_1\ldots T_{k-1}$, the start-time and the fulfill-time are $e_{A(i)}^{A(k)} = e_j^L$ and $f_{A(j)}^{A(k)} = f_i^L$, respectively for $j = 0$ to $k - 1$. The feasibility of sub-schedule $T_{A(0)}T_{A(1)}\ldots T_{A(k-1)}$ ($f_{A(j)}^{A(k)} = f_j^L \leqslant d_{A(j)} = d_j$ for $j = 0$ to $k - 1$) is guaranteed.

(b) In schedule sequence $T_xT_{A(k)}$, we can compute the minimal schedulable fulfill-time $f_x^{A(k)}$ and $f_{A(k)}^{A(k)}$ as follows:

$$f_x^{A(k)} = e_x^{A(k)} + c_{A(k-1),x} = \max\{r_x, f_{A(k-1)}^L\} + c_{A(k-1),x} \tag{9}$$

$$f_{A(k)}^{A(k)} = e_{A(k)}^{A(k)} + c_{x,A(k)} = \max\{r_{A(k)}, f_x^{A(k)}\} + c_{x,A(k)} = \max\{r_{A(k)}, \max\{r_x, f_{A(k-1)}^L\} + c_{A(k-1),x}\} + c_{x,A(k)}. \tag{10}$$

Since $f_{A(k-1)}^L$ is pre-computed, real-time requirements $f_x^{A(k)} \leqslant d_x$ and $f_{A(k)}^{A(k)} \leqslant d_{A(k)}$ can be verified in O(1) time.

(c) From the definition of schedulable-region, real-time requirements ($f_{A(j)}^{A(k)} \leqslant d_{A(j)}$ for $j = k + 1$ to n) of the remainder schedule $T_{A(k+1)}T_{A(k+2)}\ldots T_{A(n)}$ is guaranteed if $f_{A(k)}^{A(k)} \leqslant f_{A(k)}^R$. Since $f_{A(k)}^R$ is pre-computed and $f_{A(k)}^{A(k)}$ can be computed from Eq. (10), the feasibility of sub-schedule $T_{A(k+1)}T_{A(k+2)}\ldots T_{A(n)}$ can be verified in O(1) time.

(d) According to the pre-computed $f_{A(j)}^L$ and $f_{A(j)}^R$ (for $j = 0$ to n), the feasibility of rescheduled result $T_{A(0)}T_{A(1)}\ldots T_{A(k-1)}T_xT_{A(k)}\ldots T_{A(n)}$ can be verified in O(1) time. □

Fig. 6 shows an example for demonstrating the schedulable-region concept (input schedule is $T_0T_2T_1T_3$ as shown in Fig. 1). The null taskT_4 is added to the end of schedule. Assume that task T_3 is selected and removed from input schedule for rescheduling. Using our algorithm, we can reschedule task T_3 into the front of T_1.

Note that, in the proposed GSR algorithm, tasks are selected and rescheduled into the *best-fit* scan-groups to minimize the schedule fulfill-time under real-time requirements. We need to calculate not only the feasibility, but also the *schedule fulfill-time* of rescheduled result (to determine if the new rescheduled result obtains a better data throughput than other schedules). Assume that the rescheduled result is $T_{A(0)}T_{A(1)}\ldots T_{A(k-1)}T_xT_{A(k)}\ldots T_{A(n)}$ as described above. The improvement of schedule fulfill-time $v^{A(k)}_{A(n)}$ can be calculated in O(1) time.

Theorem 2. *With the pre-computed* $W_{A(i)} = \sum_{j=n}^{i+1}(e^{\mathrm{R}}_{A(j)} - f^{\mathrm{R}}_{A(j-1)})$ *and* $V_{A(i)} = min\{(e^{\mathrm{R}}_{A(j)} - f^{\mathrm{R}}_{A(j-1)} + W_{A(j)}); for\ j = n\ to\ i+1\}$, *for* $i = 0$ *to* n, *the improvement of schedule fulfill-time* $v^{A(k)}_{A(n)}$ *for the rescheduled result* $T_{A(0)}T_{A(1)}\ldots T_{A(k-1)}T_xT_{A(k)}\ldots T_{A(n)}$ *can be computed by* $v^{A(k)}_{A(n)} = \min\{V_{A(k)}, f^{\mathrm{R}}_{A(k)} - f^{A(k)}_{A(k)} + W_{A(k)}\}$ *in* $O(1)$ *time.*

Proof. Divide this rescheduled result $T_{A(0)}T_{A(1)}\ldots T_{A(k-1)}T_xT_{A(k)}\ldots T_{A(n)}$ into three sub-schedules: $T_{A(0)}T_{A(1)}\ldots T_{A(k-1)}$, $T_xT_{A(k)}$ and $T_{A(k+1)}T_{A(k+2)}\ldots T_{A(n)}$.

(a) Since the first sub-schedule $T_{A(0)}T_{A(1)}\ldots T_{A(k-1)} = T_0T_1\ldots T_{k-1}$ is not changed, we have $e^{A(k)}_{A(i)} = e^{\mathrm{L}}_i$ and $f^{A(k)}_{A(i)} = f^{\mathrm{L}}_i$ for $i = 0$ to $k-1$.

(b) Define the improvement of schedule fulfill-time for sub-schedule $T_{A(0)}T_{A(1)}\ldots T_{A(i)}$ as $v^{A(k)}_{A(i)} = f^{\mathrm{R}}_{A(i)} - f^{A(k)}_{A(i)}$. In sub-schedule $T_xT_{A(k)}$, the fulfill-time $f^{A(k)}_{A(k)}$ can be computed by Eq. (10). The improvement in schedule fulfill-time

$$v^{A(k)}_{A(k)} = f^{\mathrm{R}}_{A(k)} - f^{A(k)}_{A(k)} \tag{11}$$

is obtained in O(1) time.

(c) As shown in Fig. 7, for task $T_{A(i)}$ in the remainder sub-schedule $T_{A(k+1)}T_{A(k+2)}\ldots T_{A(n)}$, the improvement in schedule fulfill-time $v^{A(k)}_{A(i)}$ (for $i = k+1$ to n) can be computed by

$$v^{A(k)}_{A(i)} = \min\{u_{A(i)}, v^{A(k)}_{A(i-1)} + w_{A(i)}\} \tag{12}$$

where the parameters $u_{A(i)} = e^{\mathrm{R}}_{A(i)} - r_{A(i)}$ and $w_{A(i)} = e^{\mathrm{R}}_{A(i)} - f^{\mathrm{R}}_{A(i-1)}$ denote the upper bound and the lower bound of improvement $v^{A(k)}_{A(i)}$, respectively.

(d) The improvement in schedule fulfill-time $v^{A(k)}_{A(n)}$ for an arbitrary task $T_{A(k)}$ can be defined as the following recursive function.

$$v^{A(k)}_{A(n)} = \begin{cases} \min\{u_{A(n)}, v^{A(k)}_{A(n-1)} + w_{A(n)}\} & \text{if } (n > k) \\ f^{\mathrm{R}}_{A(n)} - f^{A(k)}_{A(n)} & \text{if } (n = k) \end{cases} \tag{13}$$

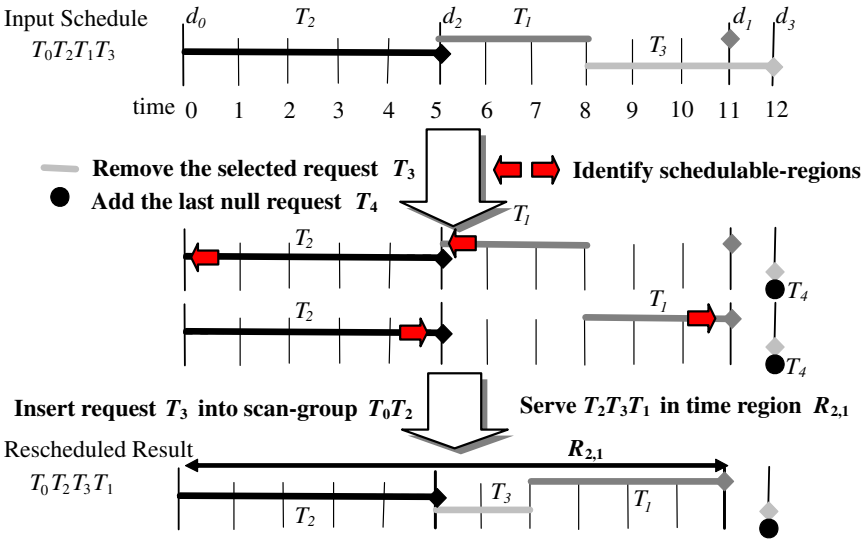

Fig. 6. A simple example to demonstrate the schedulable-region concept. Task T_3 can be served before task T_1 if $T_2T_3T_1$ can be served in the schedulable-region $R_{2,1}$.

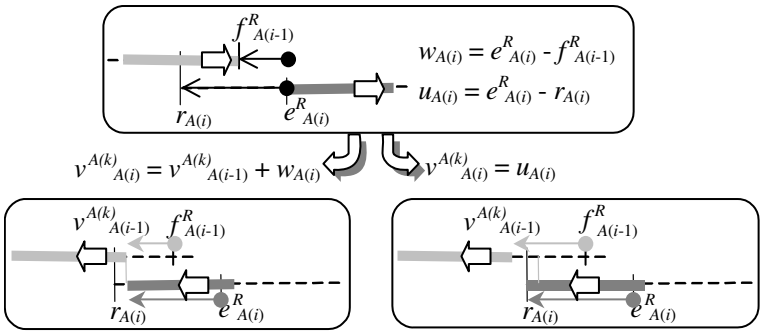

Fig. 7. A simple example to illustrate the recursive relation between the improvement $v^{A(k)}_{A(i-1)}$ and the improvement $v^{A(k)}_{A(i)}$. The upper bound and lower bound of the related improvement are shown.

Define $W_{A(i)} = \sum_{j=n}^{i+1} w_{A(j)}$, $U_{A(i)} = u_{A(i)} + W_{A(i)}$ and $V_{A(i)} = \min\{U_{A(n)}, U_{A(n-1)}, \ldots, U_{A(i+1)}\}$ for $i = 0$ to n. The above recursive function can be rewritten as follows:

$$\begin{aligned} v^{A(k)}_{A(n)} &= \min\{u_{A(n)},\ v^{A(k)}_{A(n-1)} + w_{A(n)}\} = \min\{u_{A(n)}, \min\{u_{A(n-1)}, v^{A(k)}_{A(n-2)} + w_{A(n-1)}\} + w_{A(n)}\} \\ &= \min\{u_{A(n)}, u_{A(n-1)} + w_{A(n)}, v^{A(k)}_{A(n-2)} + w_{A(n-1)} + w_{A(n)}\} \\ &= \min\{u_{A(n)}, u_{A(n-1)} + W_{A(n-1)}, \ldots, u_{A(k+1)} + W_{A(k+1)}, v^{A(k)}_{A(k)} + W_{A(k)}\} \\ &= \min\{U_{A(n)}, U_{A(n-1)}, \ldots, U_{A(k+1)}, v^{A(k)}_{A(k)} + W_{A(k)}\} = \min\{\min\{U_{A(n)}, U_{A(n-1)}, \ldots, U_{A(k+1)}\}, v^{A(k)}_{A(k)} + W_{A(k)}\} \\ &= \min\{V_{A(k)}, f^{\mathrm{R}}_{A(k)} - f^{A(k)}_{A(k)} + W_{A(k)}\}. \end{aligned} \tag{14}$$

(e) With the pre-computed $W_{A(i)}$ and $V_{A(i)}$ (for $i = 0$ to n), the improvement in schedule fulfill-time $v^{A(k)}_{A(n)}$ for an arbitrary task $T_{A(k)}$ can be obtained in O(1) time. □

As there are at most n scan-groups to be considered, thus, the GSR takes O(n) time to find out the best rescheduled result $T_{A(0)}T_{A(1)}\ldots T_{A(y-1)}T_xT_{A(y)}\ldots T_{A(n)}$ that satisfies $v^{A(y)}_{A(n)} = \max\{v^{A(k)}_{A(n)}$, for serving T_x before tasks $T_{A(y)}$ in different scan-groups$\}$.

5. Supporting non-real-time tasks

In a real-time system, although most disk accesses are timing critical, there are still a few disk tasks for non-real-time data access. For example, in a Video-on-Demand (VoD) system, users may first browse the archive to select the desired video. After that, a continuous real-time retrieval of selected video should be guaranteed by the applied real-time disk-scheduling scheme for jitter-free playback. Although the non-real-time browsing task has no deadline constraints, reasonable response time should be offered to provide a comfortable service to users. Of course, such a reasonable response must be offered under the timing requirements of real-time tasks.

Intuitively, non-real-time tasks would be served after the completion of all real-time tasks. However, such an approach would cause an undesired large response time and at worst, be starved of service to non-real-time tasks. Assume that $T_0T_1\ldots T_n$ is the original schedule and T_{n+1} is a newly added non-real-time task. Another naive approach would set the ready time $r_{n+1} = 0$ and the deadline $d_{n+1} = \infty$, thus, non-real-time task T_{n+1} can be viewed as a real-time task and scheduled by previous real-time scheduling algorithms. For example, as shown in Fig. 8, the non-real-time task will be placed at

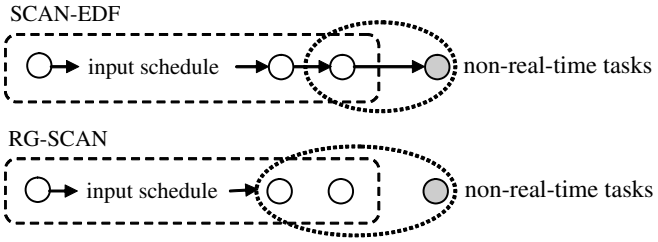

Fig. 8. Non-real-time tasks can only be rescheduled with the last task or the last scan-group in SCAN-EDF and RG-SCAN.

the end of input schedule and rescheduled with the *last* task (by SCAN-EDF) or the *last* R-Group (by RG-SCAN). Nevertheless, such an approach still results in an undesired long response time. Although we may assign an earliest deadline to the non-real-time task, the choice of a proper deadline is not easy.

In this paper, we extend the GSR algorithm described in Section 3.2 to serve fairly non-real-time tasks. Let $T_{A(0)}T_{A(1)}\ldots T_{A(n)} = T_0T_1\ldots T_n$ be the schedule of real-time tasks. The non-real-time task T_{n+1}, just as T_i described in Algorithm 2, is selected from the task queue for rescheduling into one of the scan-groups. Note that, as our previous algorithm is designed to maximize disk throughput, the *best-fit* scan-group is selected for rescheduling. As a result, $T_i = T_{n+1}$ will not be served as soon as possible. However, for serving non-real-time tasks, the response time should be minimized. Consequently, for each non-real-time task, we try to reschedule it into its ***first-fit*** scan-group. To serve T_{n+1} as soon as possible, the schedulable-region of $T_{A(0)}T_{A(1)}\ldots T_{A(n)}$ should be upper bounded by the deadline d_n of the last task T_n (it is upper bounded by f_n^L in Algorithm 2). Moreover, it is not necessary to add a null task with deadline f_n^L to the end of input schedule. By applying similar operation steps in Algorithm 2, we try to reschedule non-real-time task T_x ($=T_{n+1}$) into the *first-fit* scan-group S_k (it may not be the *best-fit* one in terms of maximizing data throughput). The index k is as small as possible to offer non-real-time task T_x a short response time without causing the real-time tasks to violate their timing requirements.

As shown in the proposed fast algorithm, after pre-computing some problem parameters $W_{A(i)}$ and $V_{A(i)}$ (for $i = 0$ to n), the feasibility for serving each rescheduled result $T_{A(0)}T_{A(1)}\ldots T_{A(k-1)}T_xT_{A(k)}\ldots T_{A(n)}$ can be verified in constant time. It takes a total of $O(n)$ time to decide the best schedule result that serves fairly non-real-time tasks without violating timing requirements of the original schedule. Its time complexity is the same as that of conventional methods.

6. Experimental results

In this section, the experimental results of our proposed GSR algorithm are compared with those of previous approaches. Table 1 shows the important parameters of HP 97560, which is used as the disk model in our experiments (Reddy and Wyllie, 1993). The seek time cost is defined in Eq. (2). The rotational latency is assumed half of the time of a full track revolution. Each real-time task is assumed to request for a track of data (36 KB in HP 97560). Ready times of tasks are uniformly distributed among 0 and 240 ms. The related deadline is the summation of its ready time and a period time that varies from 120 to 480 ms. Non-real-time tasks are assumed to arrive with a Poisson distribution. The mean inter-arrival time between each non-real-time task is described in the related experiments. The queuing principle for non-real-time tasks is followed by FIFO order for its simplicity and fairness. The size of data accessed by each non-real-time task is assumed to be 4 KB. The workloads of both real-time and non-real-time tasks are uniformly distributed over the disk surface. In all following experiments, 100 experiments are conducted with different seeds for random number generation and the average value is used for performance evaluation.

6.1. Number of supported tasks

The number of tasks supported is one of the most important factors in measuring performance of a real-time disk scheduling algorithm. In Table 2, we summarize the minimum, the maximum and the average number of real-time disk tasks that can be supported by different methods, such as GSR, RG-SCAN, and SCAN-EDF. For fair comparison, we apply the same one hundred test examples to different test methods. In each test example, a set of 30 real-time disk tasks are given and the number of feasibly completed tasks is counted.

According to the experimental results, our GSR method can support more tasks than conventional schemes. In average, the number of tasks supported by our method is larger than the conventional SCAN-EDF method with 50% improvements. Comparing to RG-SCAN, our method shows around 10% improvements. Notably, based on the above experiments, test examples that are not schedulable for conventional method can be successfully scheduled by our GSR method. This is because GSR scheme is a global seek-optimizing algorithm. As a result, compared to previous local

Table 1
Disk parameters of HP 97560

No. of cylinders per disk	1972
No. of tracks per cylinder	19
No. of sectors per track	72
Sector size	512 bytes
Seek time function (ms)	$\text{Seek(d)} = \begin{cases} 3.24 + 0.4\sqrt{d}, & d \leqslant 383 \\ 8.00 + 0.008d, & d > 383 \end{cases}$
Revolution speed	4002 RPM
Transfer time	10 MBps

Table 2
The minimal, maximal, and average number of supported real-time tasks under different scheduling policies

Algorithms	Number of supported tasks		
	Minimum	Maximum	Average
GSR	18	28	23
RG-SCAN	18	27	21
SCAN-EDF	14	23	15

seek-optimizing schemes, input tasks' services times are further reduced after rescheduling and thus more tasks can be served before their deadlines. In other words, the further reduction of tasks' service times prompts more tasks to be served.

6.2. Improvement of disk throughput

Note that, if the same input tasks are given, a well-behaved real-time disk-scheduling algorithm should finish the schedule as quickly as possible to maximize data throughput. In this paper, test workloads with different numbers of input tasks are employed to measure the disk throughput obtained under different disk-scheduling schemes. To compare the results with SCAN-EDF, we let the test workloads to be feasibly scheduled by the EDF method. The result is shown in Fig. 9. Note that the improvement in data throughput is compared with achieved by SCAN-EDF. Experiments show that the data throughput obtained by our GSR method is always better than that obtained by SCAN-EDF and DM-SCAN, regardless of the problem size of test workload. Table 3 summarizes the minimum, the maximum, the average schedule fulfill-time, and the improvement in data throughput obtained under 15 input real-time tasks for detailed comparisons. Table 3 shows that the data throughput achieved by GSR is 1.1 times that of RG-SCAN's. Table 4 presents the same performance metric but assumes 20 input real-time tasks. Likewise, our GSR scheme achieves 11% improvement when compared with the RG-SCAN scheme. As stated in Section 2, previous schemes are locally seek-optimizing scheme; a task can only be rescheduled to the *locally best* position (within a limited group) in terms of data throughput improvement. In contrast, GSR is a globally rescheduling scheme; that is, a task is rescheduled to the *globally best* position. As a result, the further reduction in service time of tasks promotes a higher disk throughput achieved by the GSR scheme.

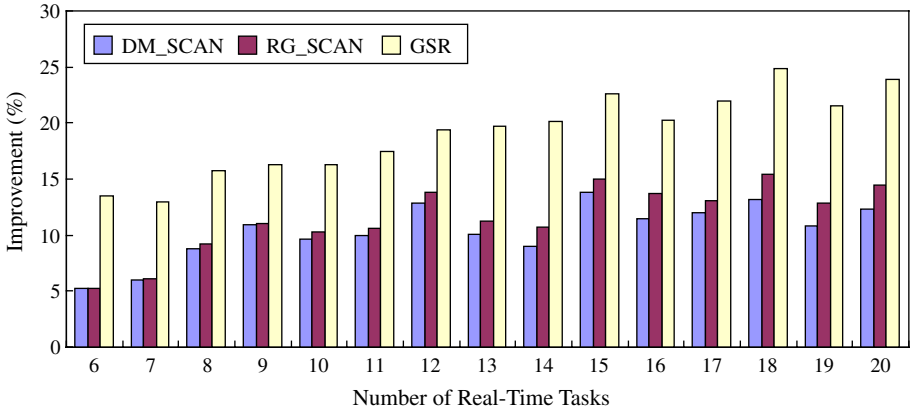

Fig. 9. The data throughput improvement of various real-time disk scheduling algorithm under different number of input real-time tasks.

Table 3
Given 15 real-time tasks, the minimum, maximum, average schedule fulfill-time, and the obtained data throughput improvement of different disk scheduling approaches

Algorithms	Schedule fulfill-time (ms)			
	Minimum (ms)	Maximum (ms)	Average (ms)	Improvement (%)
GSR	139.23	207.58	172.46	22.60
RG-SCAN	146.34	261.64	189.35	15.02
DM-SCAN	154.44	261.64	191.98	13.84
SCAN-EDF	173.73	283.82	222.82	≈0

Table 4
Given 20 real-time tasks, the minimum, maximum, average schedule fulfill-time, and the obtained data throughput improvement of different disk scheduling approaches

Algorithms	Schedule fulfill-time (ms)			
	Minimum (ms)	Maximum (ms)	Average (ms)	Improvement (%)
GSR	180.75	277.47	221.31	23.91
RG-SCAN	195.15	318.54	248.90	14.43
DM-SCAN	195.15	318.54	254.88	12.37
SCAN-EDF	236.52	338.36	290.86	≈0

Furthermore, we present the data throughput improvement under a mixed workload. In RG-SCAN, they also extend their algorithm to serve fairly non-real-time tasks. Thus, we compare our performance result with that obtained by RG-SCAN scheme. Assume that the mean inter-arrival time of non-real-time tasks is 10.1 ms. Fig. 10 shows the data throughput improvement under different scheduling schemes while each input problem size consists of a number of real-time tasks and three non-real-time tasks. For example, the bar under 10 real-time tasks actually consists of 10 real-time tasks and three non-real-time tasks. As stated before, lots of systems may consist of a mixed workload that has both real-time and non-real-time traffics. Thus, the applied disk-scheduling scheme should maximize the disk throughput under guaranteed real-time constraints while minimizing the response time to non-real-time tasks. As seen in Fig. 10, the data throughput improvement of GSR still outperforms the conventional schemes in a mixed environment. For example, given 15 real-time tasks and three non-real-time tasks, our data throughput is 1.1 times that of RG-SCAN's. Note that, we will show the effectiveness of our scheme concerning the performance of non-real-time tasks in Section 6.3. Fig. 11 shows the same performance metrics but each problem size consists of five non-real-time disk tasks.

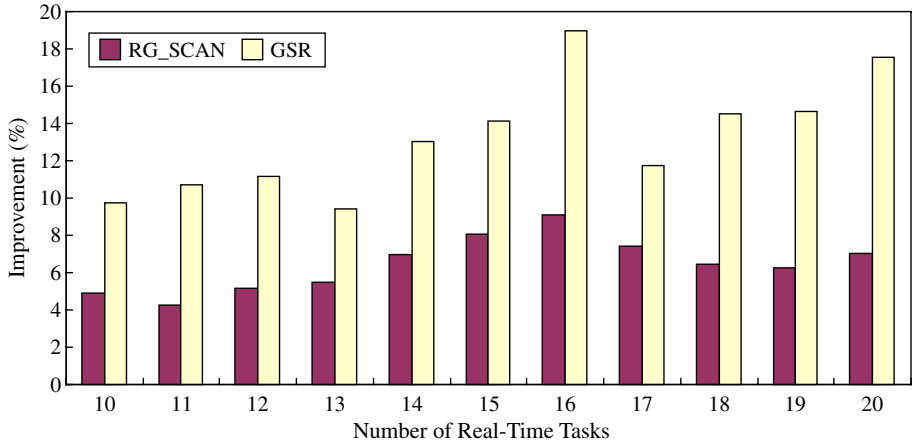

Fig. 10. Given three non-real-time tasks, the data throughput improvement of GSR and RG-SCAN under different number of input real-time tasks.

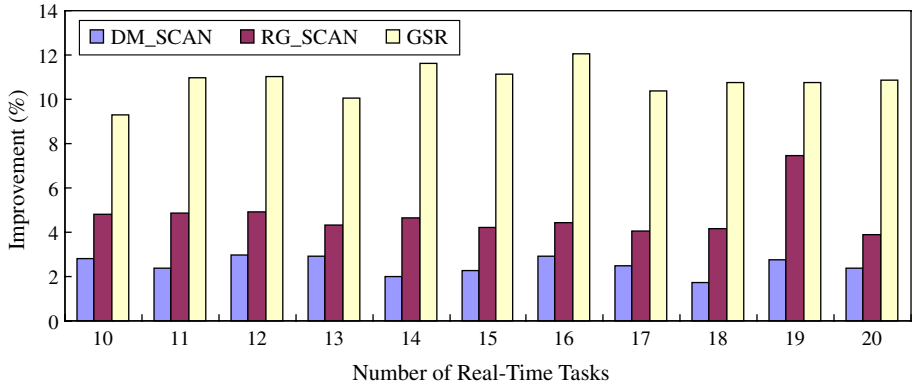

Fig. 11. Given five non-real-time tasks, the data throughput improvement of GSR and RG-SCAN under different number of input real-time tasks.

6.3. Response time of non-real-time task

A real system would consist of a mixed workload and require serving both real-time and non-real-time tasks. As a result, given a real-time schedule and a number of non-real-time tasks, the original real-time schedule should be adjusted to serve fairly the non-real-time tasks but without violating timing constraints. For a non-real-time task, the response time that counts the difference between its fulfill-time and its ready-time plays an important factor for measuring the effectiveness of a disk-scheduling algorithm. Given 10 real-time tasks, Fig. 12 shows the average response time obtained by GSR and RG-SCAN under different number of non-real-time tasks. The mean inter-arrival time of non-real-time tasks is assumed to be 10.1 ms, which saturates the queue of non-real-time task to avoid the occurrence of an empty queue. To show the effectiveness of our scheme in supporting the non-real-time tasks under different non-real-time task workload, Figs. 13 and 14 show the same performance metric but the mean inter-arrival time of each non-real-time task is set to 5.1 and 20.1 ms, respectively. Figs. 15–17 also show the experimental results but the problem size consists of 20 real-time tasks.

As stated in Section 2, RG-SCAN partitions the input tasks into a set of R-Groups. After rescheduling tasks by the seek-optimizing SCAN scheme within an R-Group, the finish-time of the R-Group is improved. As a result, a slack is derived between the advanced finish-time and the original one. RG-SCAN thus uses this slack to serve non-real-time tasks. If the slack derived in an R-Group is not large enough to sustain the execution of a non-real-time task, RG-SCAN continue to identify the next R-Group and the derived slack is added to the previous one and so on, until the non-real-time task can be served. In other words, in RG-SCAN, a non-real-time task is served after the real-time tasks until an enough slack is encountered. In contrast, the GSR treats non-real-time tasks with the same manner of the real-time task but uses the best-fit selection scheme and without real-time requirements. Thus, a non-real-time task in GSR is served in its earliest possible point as long as the schedule result is feasible. In addition, as shown in Section 6.1, owing to the superiority of our

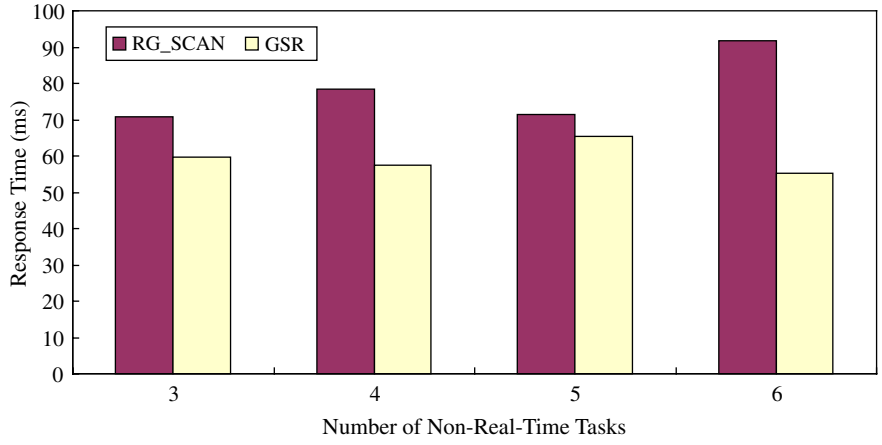

Fig. 12. Mean inter arrival time = 10.1, given 10 real-time tasks, the response time of GSR and RG-SCAN under different number of input non-real-time tasks.

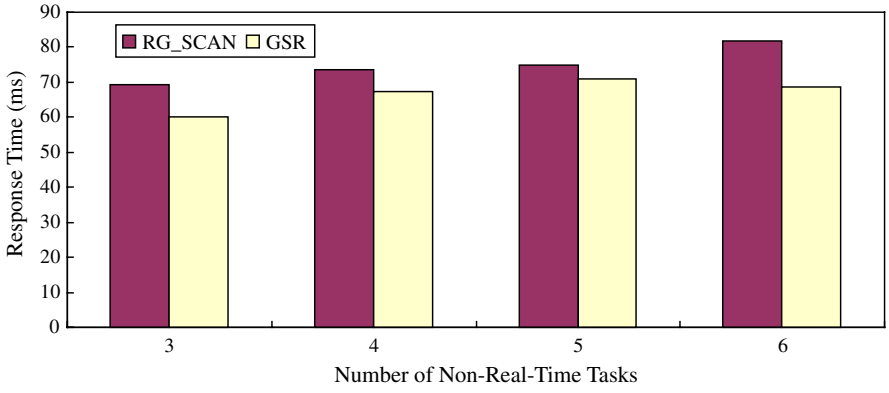

Fig. 13. Mean inter arrival time = 5.1, given 10 real-time tasks, the response time of GSR and RG-SCAN under different number of input non-real-time tasks.

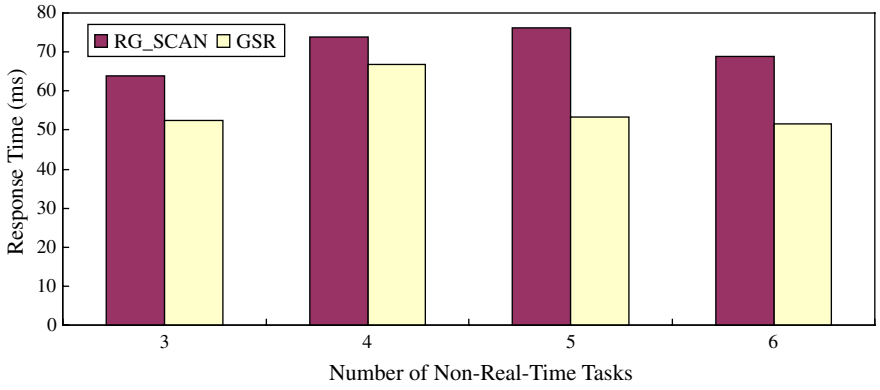

Fig. 14. Mean inter arrival time = 20.1, given 10 real-time tasks, the response time of GSR and RG-SCAN under different number of input non-real-time tasks.

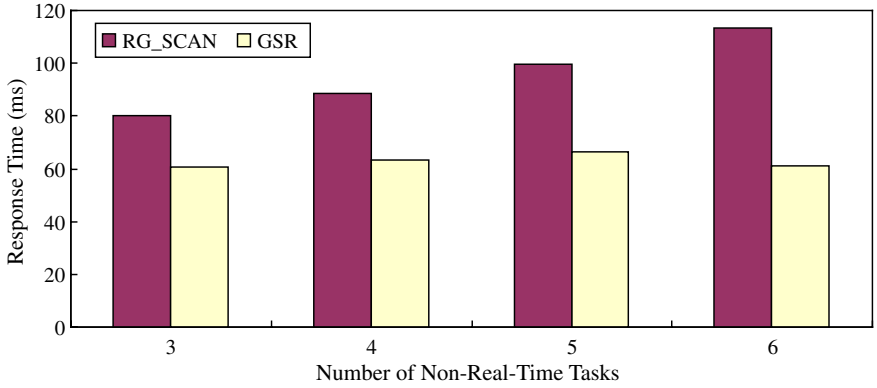

Fig. 15. Mean inter arrival time = 10.1, given 20 real-time tasks, the response time of GSR and RG-SCAN under different number of input non-real-time tasks.

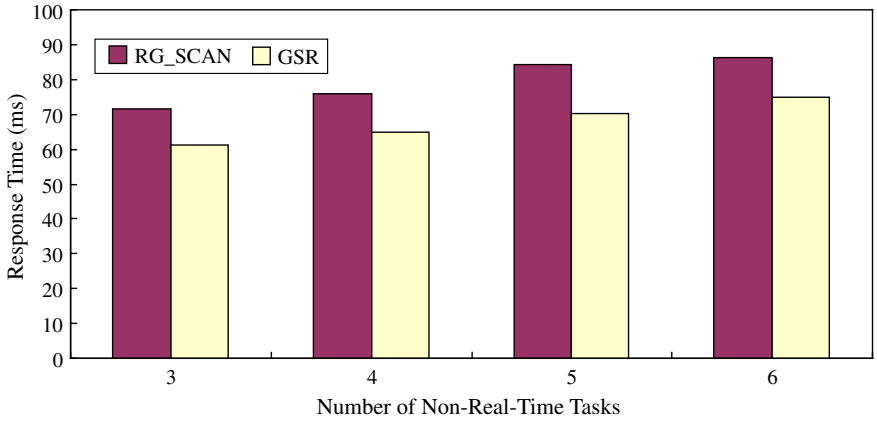

Fig. 16. Mean inter arrival time = 5.1, given 20 real-time tasks, the response time of GSR and RG-SCAN under different number of input non-real-time tasks.

proposed GSR scheme in serving real-time tasks, real-time tasks are served more quickly than by the RG-SCAN scheme. As a result, in a mixed workload, a non-real-time task can also be quickly served by the GSR scheme since real-time tasks ahead of it are soon finished. Consequently, by adapting the best-fit selection scheme to the first-fit selection criterion for non-real-time tasks and the superiority in serving real-time tasks, our proposed GSR scheme offers a shorter response time than RG-SCAN for serving non-real-time tasks.

Table 5 summarizes the minimum, maximal, and the average schedule fulfill-time and the minimum, maximal, and the average response time for non-real-time tasks for 10 real-time tasks and three non-real-time tasks with 10.1 ms mean

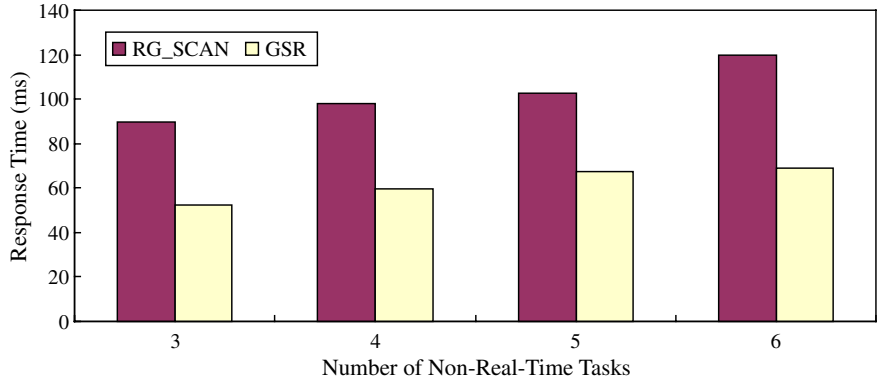

Fig. 17. Mean inter arrival time = 20.1, given 20 real-time tasks, the response time of GSR and RG-SCAN under different number of input non-real-time tasks.

Table 5
Given 10 real-time tasks and three non-real-time tasks, the schedule fulfill-time and the response time of different real-time disk scheduling approaches

Algorithm	Schedule fulfill time (ms)				Response time (ms)		
	Minimum	Maximum	Average	Improvement (%)	Minimum	Maximum	Average
GSR	136.63	184.53	164.86	9.72	13.85	113.88	59.61
RG-SCAN	134.23	195.77	173.61	4.93	19.85	114.55	70.93
DM-SCAN	151.41	204.79	177.19	2.97	19.85	114.55	70.93
SCAN-EDF	160.18	249.34	199.75	≈0	N.A.	N.A.	N.A.

Table 6
Given 20 real-time tasks and five non-real-time tasks, the schedule fulfill-time and the response time of different real-time disk scheduling approaches

Algorithm	Schedule fulfill time (ms)				Response time		
	Minimum	Maximum	Average	Improvement	Minimum	Maximum	Average
GSR	267.78	377.53	308.39	10.85	26.64	122.47	66.40
RG-SCAN	273.35	390.33	332.42	3.90	39.79	167.12	99.61
DM-SCAN	276.98	398.36	337.69	2.38	39.79	167.12	99.61
SCAN-EDF	285.41	408.92	345.92	≈0%	N.A.	N.A.	N.A.

inter-arrival time for detailed comparison. Note that, the average schedule fulfill-time includes both the execution of real-time tasks and non-real-time tasks. Table 6 shows the same performance metric but with 20 real-time tasks and five non-real-time tasks. As seen in Tables 5 and 6, our proposed GSR scheme not only offers shorter response time for non-real-time tasks, but also provides a larger data throughput (i.e., shorter schedule fulfill-time) for the total schedule results. For example, Table 6 shows that our GSR scheme achieves over 7% improvement compared with the RG-SCAN scheme in obtained data throughput but offers 33% improvement compared with the RG-SCAN in terms of average response time of non-real-time tasks. Notably, the extra non-real-time tasks supported by GSR and RG-SCAN, together with their real-time tasks, still have a shorter schedule fulfill time than SCAN-EDF, which only counts the real-time tasks, i.e., the input workload to the SCAN-EDF does not include the non-real-time tasks. This furthermore demonstrates the effectiveness of our proposed GSR scheme.

7. Conclusion

In order to improve data throughput, the seek-optimizing SCAN scheme should be employed to reschedule the input tasks as much as possible. However, previous approaches limit their flexibility and efficiency in that a task can only be seek-optimizing rescheduled with the tasks having the same deadline or within the same local group (a set of contiguous tasks). In order words, in conventional schemes, a task can only be rescheduled by SCAN with a *locally best* position. In this paper, we propose a globally seek-optimizing disk-scheduling scheme called GSR. In GSR, a task can be rescheduled to the *globally best* position, i.e., position with the maximal data throughput while guaranteeing a feasible schedule.

In addition, we extend the GSR scheme to serve mixed workloads that consist of both real-time and non-real-time traffic. Instead of the best-fit rescheduling policy for real-time tasks, the GSR reschedules a non-real-time task to the *first-fit* scan-group to minimize the response time. The experimental results show that our proposed GSR scheme is better than the conventional methods not only in improving the data throughput, but also in shortening response time.

References

Anderson, D.P., Osawa, Y., Govindan, R., 1991. Real-time disk storage and retrieval of digital audio/video data. Technical Report 1991, Department of Computer Science, University of California, Berkeley.

Anderson, D.P., Osawa, Y., Govindan, R., 1992. A file system for continuous media. ACM Trans. Computer Systems 10 (4), 311–337.

Chang, R.I., Chen, M.C., Ho, J.M., Ko, M.T., 1997. Designing the ON-OFF CBR transmission schedule for jitter-free VBR media playback in real-time networks. Proc. IEEE RTCSA, 2–9.

Chang, R.I., Shih, W.K., Chang, R.C., 1998. Deadline-modification-scan with maximum scannable-groups for multimedia real-time disk scheduling. In: Proceedings of the 19th IEEE Real-Time Systems Symposium, pp. 40–49.

Chang, H.P., Chang, R.I., Shih, W.K., Chang, R.C., 2002. Reschedulable-Group-SCAN Scheme for Mixed Real-Time/Non-Real-Time Disk Scheduling in a Multimedia System. J. Syst. Software 59 (2), 143–152.

Chen, T.S., Yang, W.P., 1992. Amortized analysis of disk scheduling algorithm V(R)*. J. Inf. Sci. Eng. 8, 223–242.

Chen, T.S., Yang, W.P., Lee, R.C.T., 1992. Amortized analysis of some disk scheduling algorithms: SSTF, SCAN, and *N*-Step SCAN. BIT 32, 546–558.

Gemmell, D.J., Christodoulakis, S., 1992. Principles of delay sensitive multimedia data storage and retrieval. ACM Trans. Information Systems 10 (1), 51–90.

Gemmell, D.J., Vin, H.M., Kandlur, D.D., Rangan, P.V., Rowe, L.A., 1995. Multimedia storage servers: a tutorial. IEEE Comput., 40–49.

Lehoczky, J.P., 1990. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: Proceedings of the Real-Time Systems Symposium, pp. 201–212.

Liu, C.L., Layland, J.W., 1973. Scheduling algorithms for multiprogramming in a hard real-time environment. J. ACM, 46–61.

Lougher, P., Shepherd, D., 1993. The design of a storage server for continuous media. Comput. J. 36 (1), 32–42.

Reddy, A.L.N., Wyllie, J., 1993. Disk scheduling in a multimedia I/O system. In: Proceedings of the ACM Multimedia Conference, pp. 225–233.

Reddy, A.L.N., Wyllie, J., 1994. I/O issues in a multimedia system. IEEE Comput., 69–74.

Ruemmler, C., Wilkes, J., 1994. An introduction to disk drive modeling. IEEE Comput., 16–28.

Stankovic, J.A., Buttazzo, G.C., 1995. Implications of classical scheduling results for real-time systems. IEEE Comput., 16–25.

Steinmetz, R., 1995. Multimedia file systems survey: approaches for continuous media disk scheduling. Comput. Commun. 18 (3), 133–144.

Wong, C.K., 1980. Minimizing expected head movement in one dimension and two dimension mass storage system. Comput. Survey 12 (2), 167–178.

Yee, J., Varaiya, P., 1991. Disk scheduling policies for real-time multimedia applications. Technical Report, Department of Computer Science, University of California, Berkeley.